Sprint 1 Challenge

Mine Sweeper

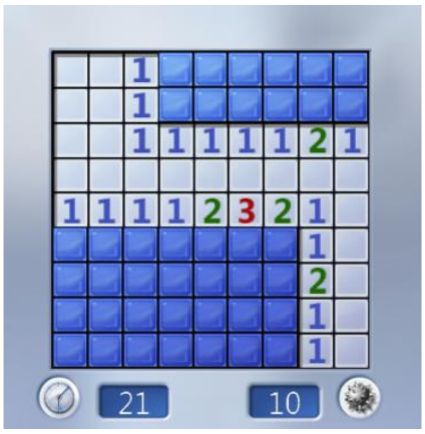
Preface

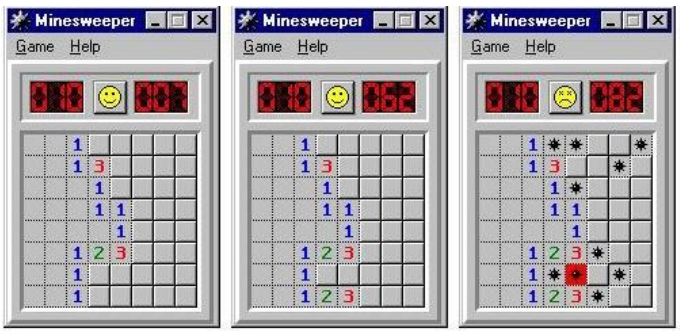
Your challenge is to create the **Minesweeper game**, and it's not an easy one. Let's practice some breaths.

Good.

Important note: Read the entire document below before start coding.

Play the game a little bit, get to know it functions and relax





It's a good thing we studied about Matrixes. Isn't it?

Software Delivery Phases - Instructions

In this sprint we will work alone, please respect the rules!

The Code shall be submitted at 3 delivery events:

1. First day 8:30pm - Partial
2. Second day 8:30pm – Final. The Sprint score will be determined by this delivery (i.e. everything shall be ready in it)
3. Saturday night 10pm – Optional. This version will not affect your sprint score however it will be reviewed and will have contribution to the overall understanding of your level.

On Sunday – all projects will be presented

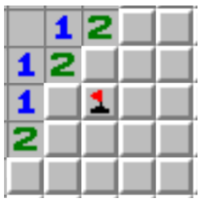
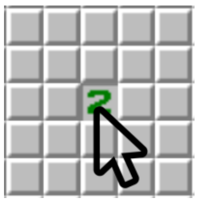
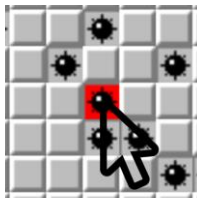
Delivery of the 3 deliveries shall be done as follows:

1. At your relevant Dropbox folder

2. Upload each delivery phase to GitHub pages and submit your project link trough a form we'll provide

Functionality and Features

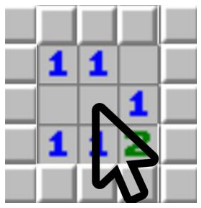
- The goal of the game is to uncover all the squares that do not contain mines without being "blown up" by clicking on a square with a mine underneath.
- Minesweeper functionality is based on the reference game
- Show a timer that starts on first click (right / left) and stops when game is over.
- Left click reveals the cell's content
- Right click to flag/unflag a suspected cell (you cannot reveal a flagged cell)
- Game ends when:
    - LOSE: when clicking a mine, all mines should be revealed
    - WIN: all the mines are flagged, and all the other cells are shown
- Support 3 levels of the game
    - Beginner (4*4 with 2 MINES)
    - Medium (8 * 8 with 12 MINES)
    - Expert (12 * 12 with 30 MINES)
- If you have the time, make your Minesweeper look great.

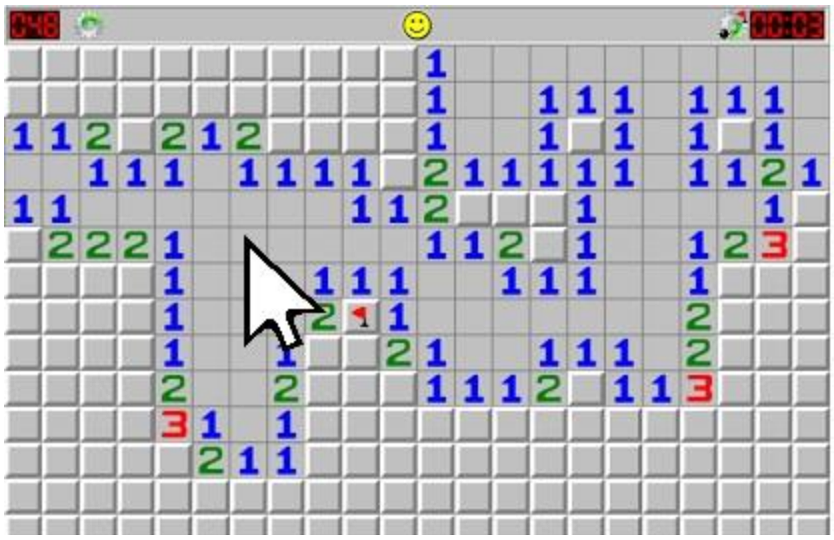
Comment regarding Bonus Tasks: implement the NON-Bonus tasks first. The Non-Bonus tasks represent the level required at the stage. Only if you have time – implement the Bonus tasks.

About Expanding – implementation guidelines

If a NON-MINE cell is clicked – open all NON-MINE 1st degree neighbors around it



Bonus (only if time permits) Full expand like in the real game:



Development - Tips and Guidelines

As you know, there is usually more than one way to approach a challenge.

But as a guideline, we suggest having the following functions (it is ok to have more functions as needed).

`initGame()`	This is called when page loads
`buildBoard()`	Builds the board Set mines at random locations Call setMinesNegsCount() Return the created board
`setMinesNegsCount(board)`	Count mines around each cell and set the cell's minesAroundCount.
`renderBoard(board)`	Render the board as a <table> to the page
`cellClicked(elCell, i, j)`	Called when a cell (td) is clicked
`cellMarked(elCell)`	Called on right click to mark a cell (suspected to be a mine) Search the web (and implement) how to hide the context menu on right click
`checkGameOver()`	Game ends when all mines are marked and all the other cells are shown

`expandShown(board, elCell, i, j)`	When user clicks a cell with no mines around, we need to open not only that cell, but also its neighbors. NOTE: start with a basic implementation that only opens the non-mine 1st degree neighbors BONUS: if you have the time later, try to work more like the real algorithm (see description at the Bonuses section below)

Here are the **globals** you might be using:

`gBoard – Matrix contains cell objects:`
 	
`{`
`    minesAroundCount: 4,`
`    isShown: true,`
`    isMine: false,`
`    isMarked: true,`
 	
`}`
	The model

`gLevel = {`
`    SIZE: 4,`
`    MINES: 2`
`};`
	This is an object by which the board size is set (in this case: 4*4), and how many mines to put

`gGame = {`
`    isOn: false,`
`    shownCount: 0,`
`    markedCount: 0,`
`    secsPassed: 0`
`}`
	This is an object in which you can keep and update the current game state:
	isOn – boolean, when true we let the user play
	shownCount: how many cells are shown
	markedCount: how many cells are marked (with a flag)
	secsPassed: how many seconds passed

Development – HOW TO START

Breaking-down the task to small tasks is a key success factor. In our case – we recommend starting from the following steps:

Step1 – the seed app:

1. Create a 4x4 gBoard Matrix containing Objects.
2. Locate 2 mines **manually**, isShown=true.
3. Present the mines using renderBoard() function.

Step2 – counting neighbors:

1. Create setMinesNegsCount() and store the numbers, isShown=true.
2. Present the board with the neighbor count and the mines using renderBoard() function.
3. Have a console.log presenting the board content – to help you with debugging

Step3 – click to reveal:

1. Make sure your renderBoard() function adds the cell ID to each cell and onclick on each cell calls cellClicked() function.
2. Make the default "isShown" to be "false"
3. Implement that clicking a cell with "number" reveals the number of this cell

Step4 – randomize mines' location:

1. **Randomly** locate the 2 mines on the board
2. Present the mines using renderBoard() function.

Next Steps: Now it's your turn to plan and implement your next steps forward.

UI Guidelines

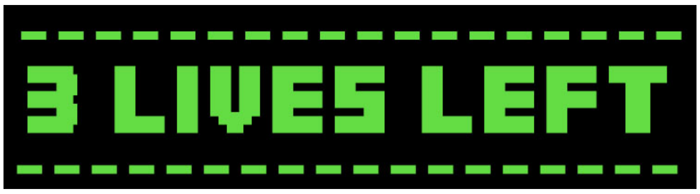
This sprint is not a UI-centered project. However, we recommend to try implementing the following UI concepts:

1. Board is square and cells are square
2. Cells keep their size when hovered and when revealed

3. Board keeps its position (shouldn't move) along all game phases (do not add UI elements dynamically above it)
4. Mines look like mines

Further Tasks

1. Make sure the first clicked cell is never a mine (like in the real game)
   HINT: place the mines and count the neighbors only on first click.
2. Add support for HINTS:
   a. The user has 3 hints:
   b. When a hint is clicked, there is an indication to the user that he can safely click one (unrevealed) cell and *reveal* it and its neighbors for a second.
   c. The clicked hint disappears.
3. Add smiley:
   - Normal
   - Sad & Dead – LOSE (stepped on a mine)
   - Sunglasses – WIN
   - Clicking the smiley – start over
4. Make the game look nice
5. Bonus: Add support for "LIVES":
   a. The user has 3 LIVES:



3 LIVES LEFT

   b. When a MINE is clicked, there is an indication to the user that he clicked a mine. The LIVES counter decrease. The user can continue playing.

6. Bonus: Keep the best score in [local storage](#) (per level) and show it on the page
7. Bonus: Full Expand

   When empty cell clicked, open all empty cells that are connected and their numbered neighbors (as is done at [the game) this feature is normally implemented using recursion.](#)

8. Bonus: Add "Safe-Click" Button:
   a. The user has a bulk of 3 "finds"
   b. Clicking the "Safe-Click" button will mark a cell (for a few seconds) that doesn't contain a MINE and has not been revealed yet.
   c. Present the remaining safe-clicks count
   d. Best if the presented "safe-click" is selected randomly (i.e. do not present the first one found by order)

9. Bonus: Manually positioned mines:
   Create a "manually create" mode in which user first positions the mines (by clicking cells) and then plays.
10. Bonus: add an "UNDO" button, each click on that button takes the game back 1 step